Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jurij Slabanja

Segmentacija in rekonstrukcija objektov iz oblaka točk z uporabo globokih nevronskih mrež

magistrsko delo

magistrski program druge stopnje
računalništvo in informatika

Mentor: prof. dr. Franc Solina

Ljubljana, 2017

Zahvala

Rad bi se zahvalil mentorju prof. dr. Francu Solini za potrpežljivost in spodbudo z vsemi zanimivimi članki.

Zahvalil bi se tudi Blažu Medenu za neizmerno pomoč pri implementaciji.

Jurij Slabanja, 2017

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
LiDAR	Light Detection And Ranging	svetlobno zaznavanje in merjenje
RANSAC	Random sample consensus	soglasje o naključnem vzorčenju
R-CNN	Region-based Convolutional Neural Network	regijska konvolucijska nevronska mreža
FRCNN	Faster R-CNN	hitrejši R-CNN
RPN	Region Proposal Network	mreža za generiranje predlog regij
MSE	Mean Squared Error	povprečna kvadratna napaka

Povzetek

Naslov: Segmentacija in rekonstrukcija objektov iz oblaka točk z uporabo globokih nevronskih mrež

Na področju računalniškega vida se je že zgodaj pojavila potreba po modeliranju vizualnih informacij s kompaktnimi modeli. Z razširitvijo vse bolj zmogljivih senzorjev za zajem vizualnih informacij je postalo to še posebej pomembno. V zadnjih nekaj letih je za hitro in učinkovito procesiranje takih informacij postala precej popularna uporaba nevronskih mrež.

V tem delu smo implementirali konvolucijsko nevronsko mrežo, s katero lahko določimo ali vsaj aproksimiramo vse objekte v oblaku točk. Začeli smo s preprosto arhitekturo za napoved parametrov enega objekta. Nato smo mrežo razširili na arhitekturo podobno Faster R-CNN, s katero lahko napovemo parametre poljubno mnogo objektov v sceni. Objekte smo modelirali s superkvadriki.

Rezultati za prvotno mrežo izgledajo precej obetavni. Za posplošeno mrežo so še vedno pretežno dobri, so pa razumljivo nekoliko slabši od prvotne mreže, saj kompleksnost problema naraste. Potrebno je segmentirati vse objekte med sabo, ne samo napovedati parametre za vsak posamezen objekt.

Ključne besede

racunalniški vid, segmentacija, 3D rekonstrukcija, oblak tock, globoke nevronske mreže, TensorFlow, Keras

Abstract

Title: Segmentation and Reconstruction of Objects from a Point Cloud with Deep Neural Networks

The need to model visual information with compact representations has existed since the early days of computer vision. With the spread of ever more powerful and capable sensors, this need has become more and more present in recent years. As such, neural networks have become the popular choice for quick and effective processing of visual data.

For this thesis we implemented a convolutional neural network with which we can determine or at least approximate all objects in a given point cloud scene. We started off with a simple architecture that could predict the parameters of a single object in a scene. Then we expanded it with an architecture similar to Faster R-CNN, that could predict the parameters for any amount of object in a scene.

The results for the initial neural network were satisfactory. The second, generalized one still gave decent results, but compared to the initial one understandably performed somewhat worse, since it was also necessary to segment all the objects apart, not just predict parameters for each one.

Keywords

computer vision, segmentation, 3D reconstruction, point cloud, deep neural networks, TensorFlow, Keras

Poglavje 1

Uvod

V zadnjem desetletju se je segmentacija objektov iz 3D oblaka točk precej razširila kot predmet raziskovanja. To je posledica tega, da so senzorji za zajem takih podatkov postali zelo kompaktni, cenovno ugodni in zmogljivi. Oblaki točk so odlični za vizualizacijo, vendar je za razumevanje scene potreben kompaktnejši zapis, kar lahko dosežemo s tem, da vsak objekt v sceni predstavimo s posamičnim modelom namesto z množico točk. V zadnjih nekaj letih je uporaba nevronskih mrež postala zelo priljubljena za tako obdelavo podatkov [1, 2]. Njihova glavna prednost je hitrost, ki je pri obilici podatkov, ki jih dobimo iz senzorjev, že skoraj nujna.

Cilj naloge je bil izdelati algoritem, v katerega podamo 3D oblak točk kot vhod, nato se s pomočjo nevronskih mrež segmentira in rekonstruira objekte v sceni. To smo dosegli z implementacijo nekoliko prirejenega FRCNN (Faster R-CNN). Glavni razliki, v katerih se naša mreža razlikuje od standardnega FRCNN, sta uporaba arhitekture SqueezeNet v skupnih plasteh FRCNN in zamenjava končnega klasifikatorja z regresijo parametrov objektov. Poleg tega je bilo potrebno oblake točk predstaviti na nek smiseln način, da se mreža lahko kaj nauči iz njih. V ta namen smo oblake točk pretvorili v globinske slike in le-te uporabili kot vhodne podatke za mrežo. Koda je na voljo na spletnem naslovu
`https://bitbucket.org/meerkqat/tf-pointcloud-classification`.

V tem poročilu naprej predstavimo področje in sorodna dela v poglavju 1.1. V poglavju 1.2 na kratko opišemo glavno uporabljeno programsko opremo. Nato v poglavju 1.3 opišemo kakšni so bili uporabljeni vhodni podatki. V poglavju 2 bolj specifično opišemo kako smo generirali podatke, ki smo jih uporabili pri učenju implementirane nevronske mreže ter na kratko opišemo še kakšne vrste objektov smo uporabili v samih scenah. V poglavju 3 opišemo našo metodologijo dela. Opisane so uporabljene arhitekture nevronskih mrež in na kratko tudi uporabljena programska oprema. V poglavju 4 podamo rezultate vsega kar smo preizkusili in implementirali. Nazadnje v poglavju 5 povzamemo naše delo in podamo še nekaj zaključnih misli.

1.1 Sorodna dela

Že zgodaj v razvoju računalniškega vida se je pokazala potreba po modeliranju zajetih vizualnih informacij z volumetričnimi modeli, ki zmanjšajo šum v informaciji in jo združijo v bolj stabilne vzorce. Eden prvih takih konceptualnih modelov so bili geoni, ki jih je predlagal Irving Biederman [3]. Nato so bili nekaj časa popularni posplošeni valji [4], vendar ker jih je bilo težko rekonstruirati iz slik, je sčasoma njihova uporaba zapadla.

Potreba po modeliranju objektov iz zajetih vizualnih informacij in specifično iz oblakov točk se je precej povečala z razširitvijo LiDAR-ja. Mnoge raziskave so poskušale preslikati urbana okolja v računalniške modele s pomočjo zračnih in talnih vozil opremljenih z LiDAR-ji [5, 6]. Uporabljeno je bilo mnogo različnih metod za segmentacijo. Nekaj primerov le-teh je določanje cilindrov v sceni z metodo RANSAC [7], prileganje primitivnih geometrijskih teles čez oblake točk [8], določanje objektov z metodo podpornih vektorjev in pomičnim 3D oknom [9], segmentacija z rezanjem grafov [10], iskanje in modeliranje superkvadrikov na podlagi paradigme gradnje in izbire [11, 12] ipd. Vse metode imajo svoje prednosti in slabosti. Na primer RANSAC je dober v tem, da ima konceptualno preprosto implementacijo in je robusten tudi, če je veliko odstopajočih točk. Prednosti prileganja superkvadrikov

sta majhno število vhodnih parametrov in robustne metode rekonstrukcije. Vendar imajo povečini vse te metode pomanjkljivost v tem, da so glede na trenutno stanje tehnologije, relativno počasne ali računsko zahtevne. Ker postajajo senzorji, ki zaznavajo okolico kot oblak točk zmeraj bolj vsesprisotni, je potrebno podatke pridobljene iz njih vse hitreje in bolj učinkovito obdelati. Iz tega razloga so se raziskave na tem področju v zadnjih nekaj letih precej usmerile v strojno učenje, specifično v nevronske mreže.

Na področju medicine se nevronske mreže že dolgo uporabljajo za določanje organov in na medicinskih slikah npr. MRI in rentgenske slike [13, 14]. Zdaj se njihova uporaba v računalniškem vidu prenaša tudi na 3D. Novejše publikacije na tem področju iščejo učinkovite rešitve za določanje 3D modelov v sceni s pomočjo nevronskih mrež [15]. Primera specifične uporabe so metode hkratne lokalizacije in mapiranja (SLAM) ter ocenjevanje človeške drže v 3D. SLAM se lahko z nevronskimi mrežami obogati tako, da se poleg čiste geometrijske reprezentacije dobljene iz senzorjev, doda še naučene oznake okolice, kar pohitri in izboljša pretekle rezultate [16, 17]. Pri ocenjevanju človeške drže, se lahko nevronske mreže naučijo, kako se sklepi relativno povezujejo med sabo, kar olajša pravilno napoved drže [18].

1.2 Keras in TensorFlow

Nevronsko mrežo smo implementirali v Pythonu s pomočjo knjižnic Keras in TensorFlow. Keras je visoko nivojski vmesnik za namensko programiranje (ang. API) nevronskih mrež, ki deluje na vrhu knjižnice TensorFlow oz. jo lahko uporablja za osnovo. Priljubljen je, ker omogoča hitro in enostavno prototipiranje konvolucijskih in rekurenčnih nevronskih mrež in zelo enostavno omogoča računanje tudi na grafičnih karticah.

TensorFlow je bolj splošno namenska programska oprema za numerično računanje z uporabo grafov podatkovnih tokov. Vozlišča v grafu predstavljajo matematične operacije, v našem primeru plasti nevronske mreže npr. konvolucije. Povezave med vozlišči so predstavljene z več dimenzijskimi po-

datkovnimi nizi oz. tenzorji, ki se prenašajo iz enega vozlišča do drugega. TensorFlow tudi podpira dokaj enostavno distribucijo na različnih platformah, ne samo na standardnih procesnih enotah in grafičnih karticah, ampak tudi na mobilnih napravah. Razvit je bil v podjetju Google predvsem za uporabo z nevronskimi mrežami, vendar je dovolj splošen da podpira tudi druge aplikacije.

1.3 Vhodni podatki

Vhodni podatki za algoritem so bili 3D oblaki točk. Da se nevronska mreža lahko iz tega kaj nauči, je potrebno niz nepovezanih točk oblikovati v neko smiselno predstavitev. V ta namen smo oblak točk preoblikovali v globinske slike, preden smo ga podali kot vhod v mrežo. Specifično smo vsako sceno pretvorili v šest globinskih slik, zajetih iz različnih pogledov.

Implementirali smo tudi možnost pretvorbe oblaka točk v vokselno predstavitev, vendar smo se potem bolj usmerili v učenje z globinskimi slikami. Razlog za to je bil boljša dokumentacija in primeri za nevronske mreže, ki uporabljajo 2D podatke, v primerjavi s 3D podatki. Večina nevronskih mrež dandanes na tem področju uporablja 2D slike za vhodne podatke. Če bi imeli več časa oz. za nadaljnje delo, bi bilo zagotovo zanimivo raziskati tudi drugo različico; ali se mreža lahko bolje nauči parametrizacijo objektov iz vokselizacije namesto iz globinskih slik.

Poglavje 2

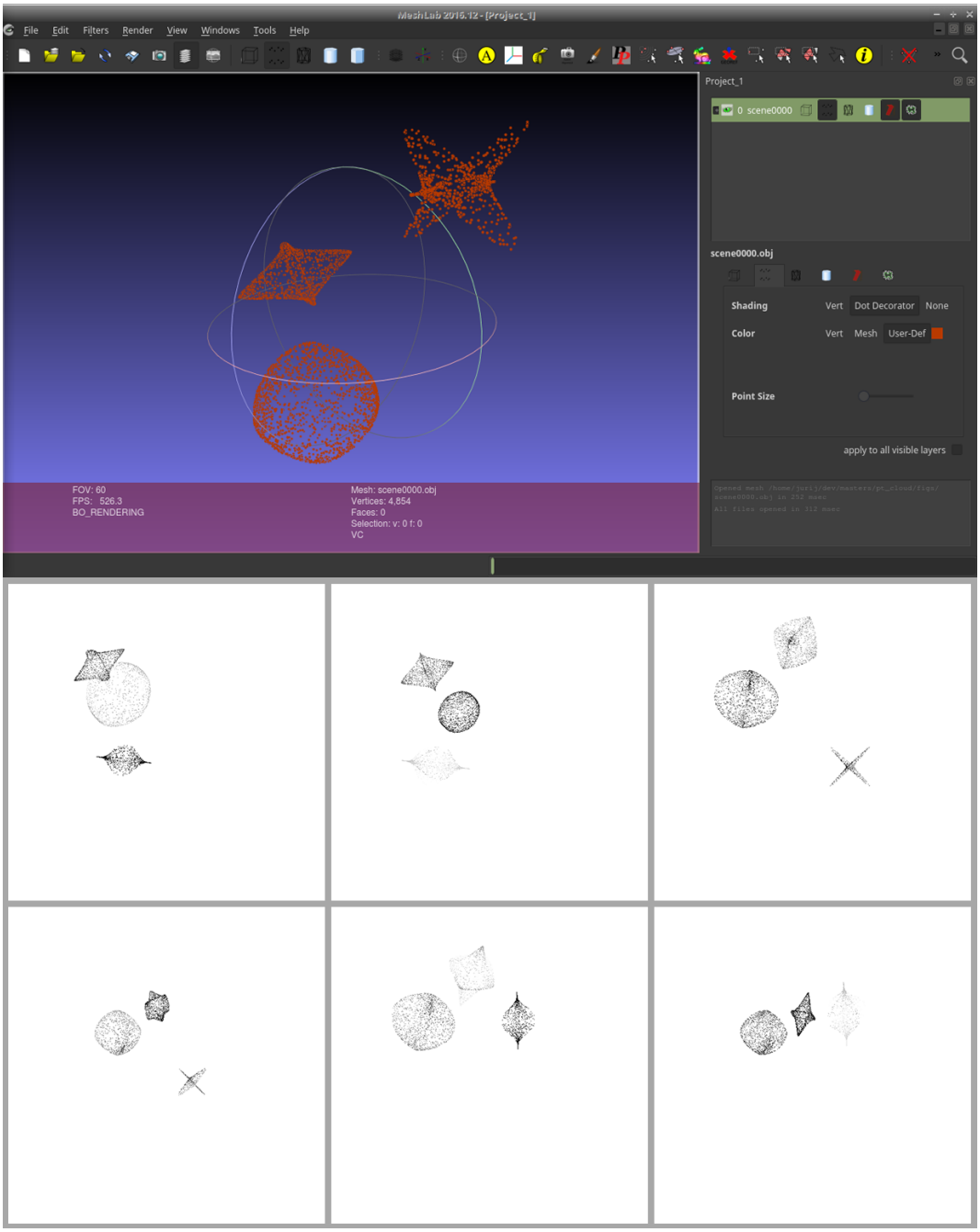
Podatkovna baza

V tem poglavju bomo opisali bazo podatkov, ki smo jo uporabili pri izdelavi naloge. Najprej bomo v poglavju 2.1 opisali kako smo ustvarili scene na katerih smo učili nevronske mreže. Nato bomo v poglavju 2.2 bolj podrobno opisali še superkvadrike, s katerimi smo modelirali objekte v scenah.

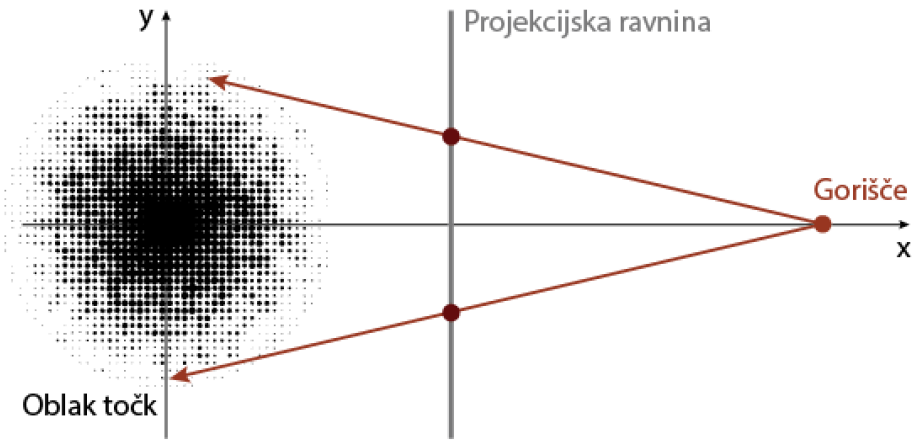
2.1 Generiranje učnih podatkov

Učni podatki za nevronsko mrežo so bili oblaki točk, predelani v globinske slike, prikazano na Sliki 2.1. Podatke smo ustvarili s po meri narejeno skripto napisano v Pythonu, ker nam ni uspelo najti obstoječe podatkovne baze, ki bi ustrezala našim specifičnim zahtevam. Za vsako sceno se je naprej naključno določilo parametre za nekaj objektov ter položaj in rotacijo vsakega objekta v sceni. Nato se je na površini vsakega objekta določilo okoli 1000 - 1500 naključnih točk. Te točke na vsakem objektu vse skupaj tvorijo celoten oblak točk scene.

Ker se iz nepovezanih točk nevronska mreža težko kaj nauči, je potrebno oblak točk na smiseln način predelati v predstavitev, iz katere se bo mreža lažje učila. V ta namen se je sceno pretvorilo v globinske slike tako, da se je pravokotno na vsako os koordinatnega sistema postavilo projekcijsko ravnino in z metodo metanja žarkov (ang. raycasting) določilo globino oz.



Slika 2.1: Na sliki je prikazan primer naključne scene. Zgoraj je oblak točk vizualiziran s programom MeshLab. Spodaj je prikazanih šest globinskih slik, ki jih generiramo iz oblaka točk.



Slika 2.2: Pretvorba oblaka točk v globinsko sliko z metodo metanja žarkov.

barvo vsake točke in kje na projekcijski ravnini leži, kot prikazuje Slika 2.2. Na koncu smo iz tega dobili šest globinskih slik dimenzij 600×600 slikovnih pik za vsako sceno, ki smo jih nato zaporedno eno za drugo vnesli v nevronsko mrežo kot vhodne podatke. Generirali smo šest slik za vsako sceno, ker nismo uporabili nobenega dodatnega predprocesiranja. Če bi vedno samo iz enega pogleda generirali sliko, bi se pogosto lahko zgodilo, da objekti ne bi bili dobro ločeni med sabo, kar bi lahko negativno vplivalo na učenje mreže.

2.2 Superkvadriki

Objekti, ki smo jih generirali za učne podatke so bili superkvadriki. Sprva smo uporabljali primitivna geometrijska telesa (kocke, krogle, valji), vendar smo na koncu prešli na superkvadrike, ker precej poenostavijo implementacijo nevronske mreže. S superkvadriki je možno modelirati mnoge različne oblike, npr. kocke, oktaedre in valje z zaobljenimi ali ostrimi robovi. To nam omogoča, da namesto mnogo različnih nevronskih mrež za napoved parametrov kock, krogel itd., uporabimo samo eno mrežo za napoved parametrov superkvadrikov in pri tem ne izgubimo raznolikosti objektov, ki jih lahko modeliramo. Prav tako se mreža ne potrebuje učiti vrste objektov oz. ne

potrebujemo posebnega modula v mreži za vsako vrsto objekta, ker so leti zmeraj superkvadriki in lahko mreža za katerikoli objekt zmeraj napove točno šest parametrov.

Splošna enačba, ki opiše vse superkvadrike v 3D je definirana s šestimi spremenljivkami na sledeči način

$$\left|\frac{x}{A}\right|^r + \left|\frac{y}{B}\right|^s + \left|\frac{z}{C}\right|^t \leq 1, \tag{2.1}$$

kjer A, B in C določijo raztege po posameznih oseh, r, s in t so pa pozitivna realna števila, ki določijo značilnosti superkvadrika. Eksponenti med 0 in 1 določajo oktaeder s konkavnimi lici in ostrimi robovi, točno 1 določajo pravilni oktaeder, med 1 in 2 določajo oktaeder s konveksnimi lici in zaobljenimi robovi, točno 2 določajo kroglo in nenazadnje bolj ko se eksponenti približajo neskončnosti, bolj je superkvadrik podoben kocki. Če so eksponenti negativni se objekt razteza v neskončnost in spada med superhiperboloide. Različne kategorije eksponentov se lahko kombinira med sabo, da dobimo kombinirane objekte. Na primer če so $r = s = 2$ in $t = 1$, prerez superkvadrika skozi xy ravnino izgleda kot krog, celoten objekt pa izgleda pretežno kot krogla s priostrenimi konicami na z osi.

Poglavje 3

Metodologija

V tem poglavju bomo opisali podrobnosti implementacije nevronske mreže, ki smo jo izdelali. V poglavju 3.1 najprej opišemo prvo nevronsko mrežo, ki smo jo implementirali. Nato v poglavju 3.2 opišemo arhitekturo osnovne nevronske mreže, katero smo uporabljali skozi celoten nadaljnji postopek izdelave naloge. Nazadnje v poglavju 3.3 opišemo arhitekturo Faster R-CNN ter se v podpoglavju 3.3.1 in podpoglavju 3.3.2 bolj specifično posvetimo RPN in regresijskemu deloma FRCNN.

3.1 Začetni CNN

Prva iteracija nevronske mreže za napoved parametrov je bila zelo preprosta. Sestavili smo jo z namenom za napoved parametrov enega objekta na sceno, centriranega na koordinatno izhodišče. Ker smo na začetku uporabljali preprosta geometrijska telesa namesto superkvadrikov, je bilo možno pred učenjem nastaviti število izhodnih parametrov. Tako smo z mrežo lahko napovedali poljubno število parametrov in je mreža lahko napovedala ne samo parametre vsakega posameznega objekta, ampak tudi lego v sliki oz. sceni, če objekt ni bil centriran. Na primer za centrirano kroglo smo lahko napovedovali samo radij, za kroglo v splošni legi pa še x, y in z koordinate.

Nadgradnja z arhitekturo FRCNN je bila potrebna, ker smo s to preprosto

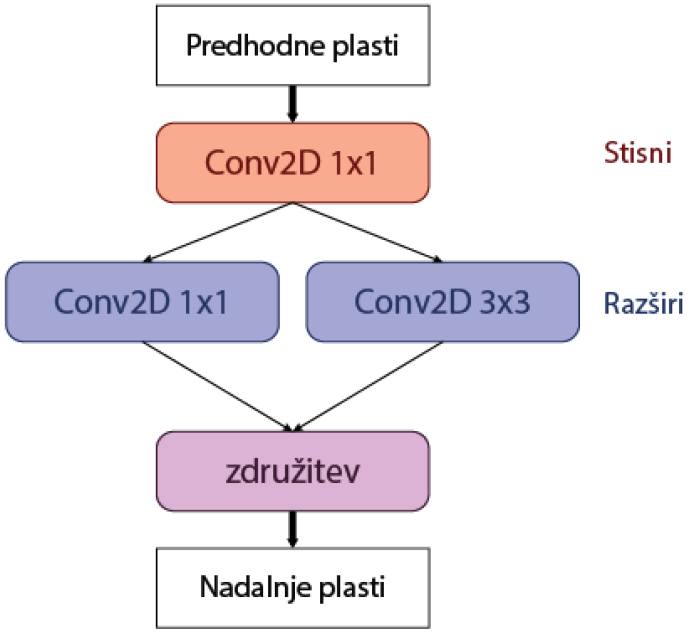
mrežo lahko parametrizirali samo en objekt v sceni oz. obravnavali sceno kot da vsebuje samo en objekt. Z mrežo za iskanje predlogov regij pa lahko sceno posplošimo na poljubno mnogo objektov.

Vhodni podatki za začetno mrežo so bile globinske slike oblaka točk, izhodni podatki pa parametri objekta v sceni. Preizkusili smo mnoge različne kombinacije konvolucijskih in polno povezanih plasti z različnimi začetnimi parametri. Prav tako smo preizkusili par standardnih arhitektur mrež (VGG16 in SqueezeNet) ter različne optimizacijske funkcije, ki jih ponuja Keras. Kar smo preizkusili je bolj točno prikazano v dodatku A. Kakšne rezultate smo dosegli s posamezno kombinacijo je bolj podrobno opisano v poglavju 4.0.1. Najboljše rezultate smo dobili z uporabo SqueezeNet, tako da smo to arhitekturo potem uporabili tudi naprej pri FRCNN.

3.2 SqueezeNet

Glavna ideja SqueezeNeta je, da se z relativno malo parametri doseže rezultate, ki so primerljivi z večjimi mrežami npr. AlexNet in VGG [19]. To doseže na podlagi treh hipotez. Naprej lahko zmanjšamo mrežo z uporabo manjših filtrov velikosti 1×1 namesto 3×3 v konvolucijskih plasteh. Tako zmanjšamo mrežo za devetkrat in namesto da filtri iščejo informacijo v okolici slikovnih pik, dejansko iščejo informacijo v posameznih kanalih slikovne pike. Nato zmanjšamo velikost vhodnih podatkov za plasti, ki še imajo filtre velikosti 3×3. To dosežemo s t.i. ognjenimi moduli (ang. fire module), ki najprej zmanjšajo oz. stisnejo število filtrov z 1×1 konvolucijo, nato pa število filtrov spet raztegnejo s kombinacijo 1×1 in 3×3 filtrov. Ta modul je osnovna gradbena enota SqueezeNeta in je prikazan na Sliki 3.1. Nazadnje vzorčenje zmanjšamo šele pozneje v mreži, tako da imajo konvolucijske plasti velike slike aktivacij (ang. activation map). To je v ravno obratno pri mnogih standardnih arhitekturah npr. VGG, vendar v članku [19] pokažejo, da kasnejše zmanjšanje vzorčenja izboljša natančnost mreže.

Celotna arhitektura SqueezNeta se začne s konvolucijsko plastjo in zliva-



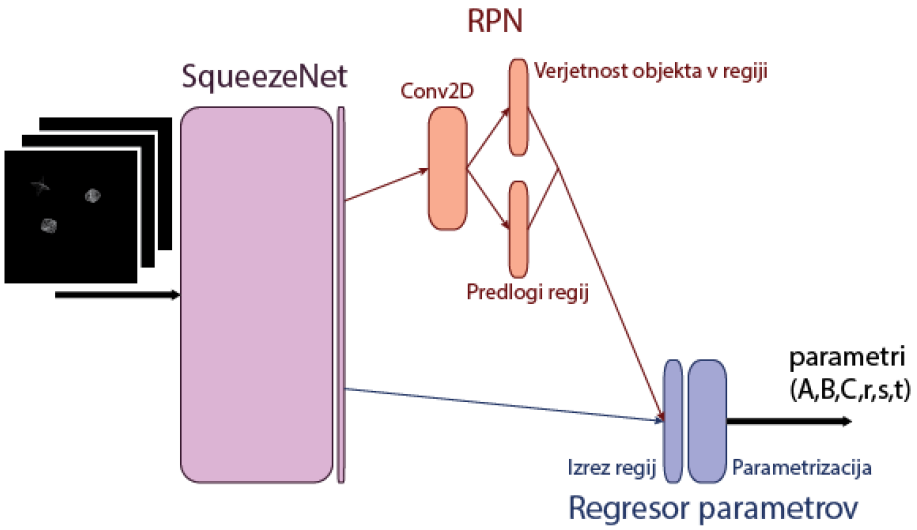
Slika 3.1: Modul, ki predstavlja osnovno gradbeno enoto arhitekture SqueezNet.

njem maksimalnih vrednosti (ang. max pooling), nato sledi zaporedje povezanih modulov prikazanih na Sliki 3.1, katerih število filtrov stalno narašča. Zaporedje modulov je na parih mestih prekinjeno še z dodatnim zlivanjem maksimalnih vrednosti, ki zmanjša slike značilk (ang. feature map).

Ker je SqueezeNet relativno majhna mreža v primerjavi z drugimi standardnimi arhitekturami, kot je VGG, je tudi zelo uporaben pri hitrem prototipiranju. Hitreje se lahko obdela učno množico in posledično zmanjša čas med učenjem in testiranjem mreže ter razvojem in razhroščevanjem.

3.3 Faster R-CNN

Napoved parametrov za enostavne scene je za preprosto nevronsko mrežo dokaj trivialna. Stvari postanejo bolj zapletene, ko imamo več objektov v poljubni legi v posamezni sceni. Potrebno je ugotoviti kje se vsak objekt



Slika 3.2: Končna arhitektura implementirane nevronske mreže za napoved parametrov objektov iz oblaka točk.

nahaja in za vsakega določiti parametre. V ta namen smo uporabili malo prilagojeno arhitekturo FRCNN [20].

Izvorna implementacija FRCNN je sestavljena iz mreže za generiranje predlogov regij (RPN) in mreže za klasifikacijo objektov v regijah. Glede na to, da nas ni zanimal tip, ampak parametri objekta, smo nadomestili klasifikacijski del z regresijo parametrov. Oba dela FRCNN si tudi delita nekaj začetnih plasti. V članku [20] opišejo uporabo arhitekture VGG16 za skupne plasti, vendar smo mi uporabili SqueezeNet, ker je dosegel vsaj primerljive, če ne boljše rezultate pri parametrizaciji objektov v prvi iteraciji naše nevronske mreže. Poleg tega je potekalo učenje SqueezeNeta mnogo hitreje kot učenje VGG16. Celotna arhitektura, ki smo jo implementirali, je prikazana na Sliki 3.2.

Ker je mreža sestavljena iz dveh delov, ki si delita skupne plasti, učenje mreže poteka v dveh korakih. V vsaki iteraciji se najprej umeri uteži v mreži na podlagi rezultatov RPN dela, nato se te rezultate uporabi za umeritev uteži v mreži na podlagi rezultatov regresijskega dela. Bolj natančno, v mrežo vsako iteracijo učenja podamo kot vhod eno globinsko sliko. Nato se izvede

RPN del in se popravi uteži v mreži glede na to koliko se rezultat iz RPN razlikuje od pričakovanega. Rezultat RPN so predlogi regij, ki določajo kje se nahajajo objekti. Nastali predlogi regij se nato uporabijo pri regresijskem delu mreže. Iz končne slike značilk skupnih plasti se izreže vse regije, se jih zmanjša oz. poveča na velikost 9×9 in se jih uporabi za učenje regresije parametrov. V vsaki iteraciji učenja se torej uteži v skupnih plasteh umerjajo na podlagi rezultatov RPN ter hkrati tudi rezultatov regresije.

Regije pridobljene iz RPN se lahko uporabijo tudi za določanje lege objekta v sceni. Če vemo kako je bila scena projecirana na projekcijske ravnine, kot je prikazano na Sliki 2.2, lahko za vsak pogled določimo premico, ki gre skozi gorišče in središče regije, ki določa nek objekt. nato za vse te premice poiščemo skupno presečišče oz. točko, ki je najbližja presečišču. V tej točki se nahaja objekt. V splošnem primeru se premice ne sekajo, ker napoved regij ni absolutno natančna. V tem delu nam časovno ni zneslo, da bi izračunali tudi to. To pomeni da sicer poznamo položaj objektov na globinskih slikah, v globalnih koordinatah scene pa ga lahko kvečjemu približno ocenimo.

3.3.1 Mreža za generiranje predlogov regij

Vhodni podatki za RPN so globinske slike generirane na podlagi 3D oblaka točk. Izhodni podatki so predlogi regij, ki določajo kje na globinskih slikah se nahajajo objekti iz oblaka točk ter ocena gotovosti, da se v določeni regiji nahaja nek objekt.

Po skupnih plasteh RPN vsebuje konvolucijsko plast, ki še dodatno obdela končno sliko značilk iz skupnih plasti ter dve končni konvolucijski plasti, ena napove regije, druga napove gotovosti objektov v regijah. Arhitektura je prikazana na zgornjem delu Slike 3.2.

Iz zadnjih dveh plasti RPN dobimo dva tenzorja, ki imata enako širino in višino kot končna slika značilk iz skupnih plasti. Globina tenzorja, ki predstavlja gotovost objektov je 9, globina tenzorja, ki predstavlja regije pa $9 \times 4 = 34$. Dimenzije tenzorjev so takšne ker za predloge regij uporabljamo

t.i. sidra (ang. anchors), to so obrobe oz. regije vnaprej določenih velikosti. Vsako sidro je določeno s štirimi parametri - x in y koordinati središča ter širina in višina sidra. Specifično uporabljamo tri resolucije sider (64, 128, 256 slikovnih pik) in tri razmerja širin in višin (1:1, 1:2, 2:1), tako da dobimo devet različnih sider, najmanjšega velikosti 64×64 in največjega 256×512 slikovnih pik. Tako kot v članku [20] dimenzije sider niso specifično umerjena za neko določeno zbirko podatkov. Tako vsaka rezina izhodnih tenzorjev RPN predstavlja eno velikost sidra. Uporaba sider naslovi problem različnih velikosti objektov na vhodnih slikah. Namesto da postopoma zmanjšujemo vhodno sliko in uporabljamo slikovne piramide ali piramide filtrov, uporabljamo piramido sider.

Ko učimo mrežo je potrebno izhodne tenzorje iz RPN malo obdelati preden jih lahko podamo v regresijski del mreže, glede na to da le-ta vsebujeta podatke o vseh možnih regijah, nas pa zanimajo samo tiste, ki tesno objemajo objekte na sliki. Da najdemo ustrezne regije, primerjamo razmerje preseka in unije med vsemi predlaganimi regijami in pravilnimi regijami, ki jih dobimo iz oznak v učnih podatkih. Bližje ko je to razmerje 1, bolj popolno se predlagana in pravilna regija prekrivata. V nasprotnem primeru če je razmerje 0, potem regiji nimata skupnega preseka. Iz tega dobimo veliko predlogov regij na katerih izvedemo zatiranje vrednosti, ki niso maksimalne (ang. non-maximum suppression), da zmanjšamo število predlaganih regij in obdržimo samo tiste, ki najbolj gotovo vsebujejo nek objekt.

Prav tako uporabimo razmerje preseka in unije, da povežemo predlagane regije z označenimi parametri objektov. Vsaka globinska slika je označena z množico parov regij in parametrov objektov na sliki. Tako predlaganim regijam lahko določimo parametre na podlagi tega s katero pravilno regijo se najbolje ujemajo. Regresijski del lahko nato učimo s predlaganimi regijami in njihovimi ustreznimi parametri.

3.3.2 Napoved parametrov

Vhodni podatki regresijskega dela so poleg globinske slike tudi predloge regij iz RPN. Izhodni podatki so parametri objekta na globinski sliki določen s specifično regijo. Po skupnih plasteh ima regresor po meri narejeno plast, ki iz končne slike značilk izreže regije dobljene iz RPN in jih raztegne ali skrči na ustrezno velikost. Vsaka regija gre nato skozi štiri polno povezane plasti, od katerih zadnja napove parametre objekta. Arhitektura je prikazana na spodnjem delu Slike 3.2.

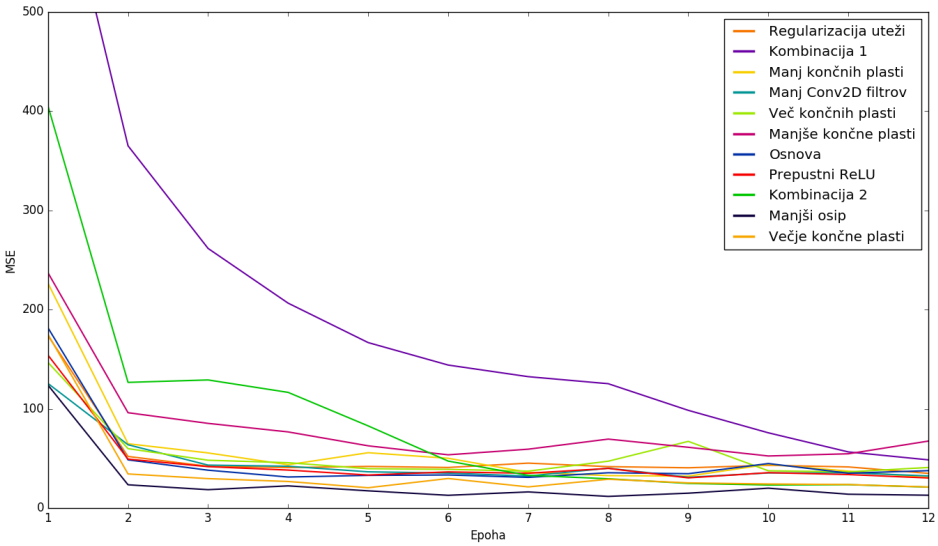
Poglavje 4

Rezultati

V tem poglavju ovrednotimo uspešnost implementiranih nevronskih mrež. Vse teste in učenje smo poganjali na grafični kartici GTX 1080 Ti z 11GB video RAMa. Najprej v poglavju 4.0.1 predstavimo rezultate prve iteracije nevronske mreže. Nato v poglavju 4.0.2 predstavimo še rezultate končne implementacije FRCNN.

4.0.1 En objekt v vsaki sceni

Prva iteracija nevronske mreže je bila namenjena napovedi parametrov iz globinskih slik centriranih objektov. Podatkovna baza za to iteracijo je bila sestavljena iz 100 učnih in 30 testnih scen. Vsaka scena je bila predstavljena s šestimi globinskimi slikami. Mrežo smo učili 12 epoh. Objekti v scenah so bile krogle centrirane na koordinatno izhodišče, kar pomeni da je mreža napovedovala radij krogle oz. en parameter. Z bolj kompleksnimi objekti in več napovedanimi parametri natančnost napovedi nekoliko pade. Preizkusili smo precej različic mreže npr. z manjšim izpustom (ang. dropout), regularizacijo uteži, različnimi velikostmi končnih polno povezanih plasti ipd. Izhodiščna konfiguracija mreže ter različne variacije so bolj točno prikazane v dodatku A. Povprečna kvadratna napaka po vsaki epohi učenja je prikazana na Sliki 4.1. Poleg tega smo preizkusili tudi različne optimizacijske funkcije, prikazano na Sliki 4.2, ter arhitekturi VGG16 in SqueezeNet, prikazano na



Slika 4.1: Povprečna kvadratna napaka po epohah za različne variacije nevronske mreže.

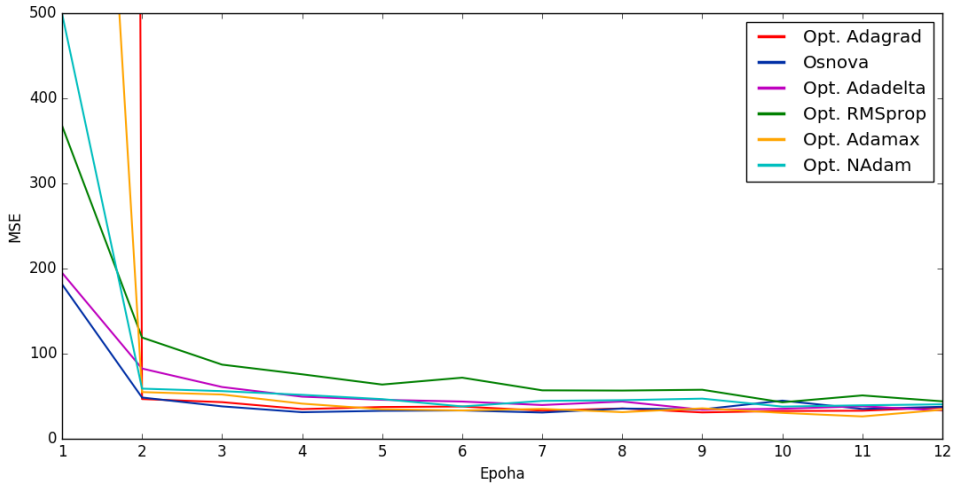
Sliki 4.3.

Točnost mreže smo merili s povprečno kvadratno napako (MSE), izračunano po sledeči enačbi

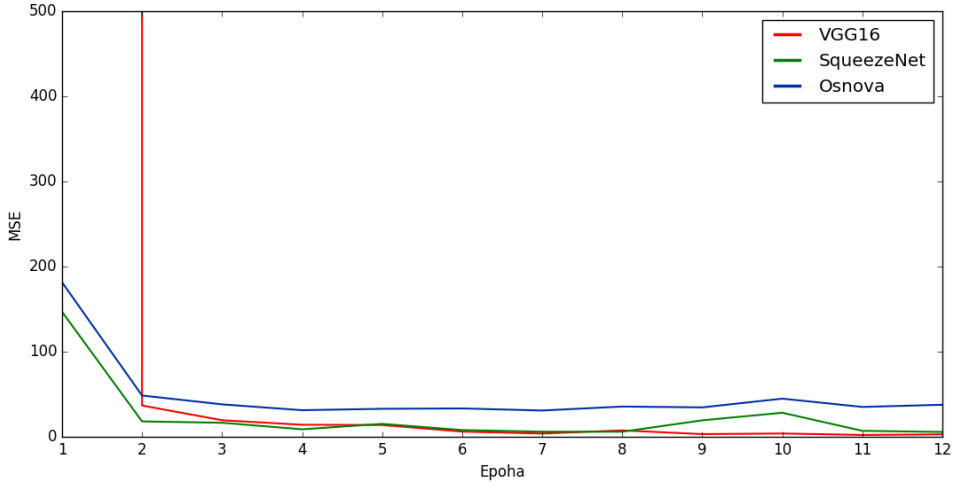
$$MSE = \frac{\sum_{j=1}^{N} \sum_{i=1}^{n} (y_i - p_i)^2}{N}. \tag{4.1}$$

V splošnih pojmih ta napaka predstavlja povprečno kvadrirano razdaljo med napovedanimi in pravilnimi parametri. V enačbi N predstavlja število napovedanih parametrov na posamezno iteracijo učenja, n dimenzijo parametrov, y_i i-to komponento napovedanega parametra in p_i i-to komponento pravilnega parametra. Dimenzija parametrov je bila $n = 1$, saj smo tu napovedovali samo radij krogle. V primeru, da bi napovedovali parametre superkvadrikov, bi bila ta dimenzija $n = 6$.

Variacije mreže vključno z izhodiščno mrežo so povečini dosegle povprečno kvadratno napako v okolici 30-40. Najboljši različici sta bili uporaba večjih končnih plasti in manj izpusta, kar kaže na to, da se z večjimi polno pove-



Slika 4.2: Povprečna kvadratna napaka po epohah za različne optimizacijske funkcije.



Slika 4.3: Povprečna kvadratna napaka po epohah za različne arhitekture nevronske mreže.

zanimi plastmi mreža lahko več nauči, z manjšim izpustom pa izgubi manj informacij. Vendar je pri tem potrebno paziti, da izpusta ne nastavimo prenizko, sicer se lahko zgodi, da v napovedi uvedemo šum.

Najslabše sta se odrezali prva kombinirana različica in uporaba manjših končnih plasti. Kombinirana različica je najverjetneje slabo delala, ker kljub temu da smo dodali polno povezano plast, smo jih hkrati nekaj tudi zmanjšali. Poleg tega smo tu tudi preizkusili manjše število filtrov na zadnji konvolucijski plasti.

Rezultati optimizacijskih funkcij so bili vsi zelo primerljivi. Razlika je bila predvsem v tem kako hitro katera funkcija konvergira.

Kar se tiče različnih arhitektur, sta se precej bolj izkazala SqueezeNet in VGG16, kot naša po meri narejena mreža. Med sabo sta bili dokaj primerljivi, razlika je bila predvsem v času izvajanja. Z uporabo SqueezeNeta je bilo učenje izvedeno mnogo hitreje kot z uporabo VGG16.

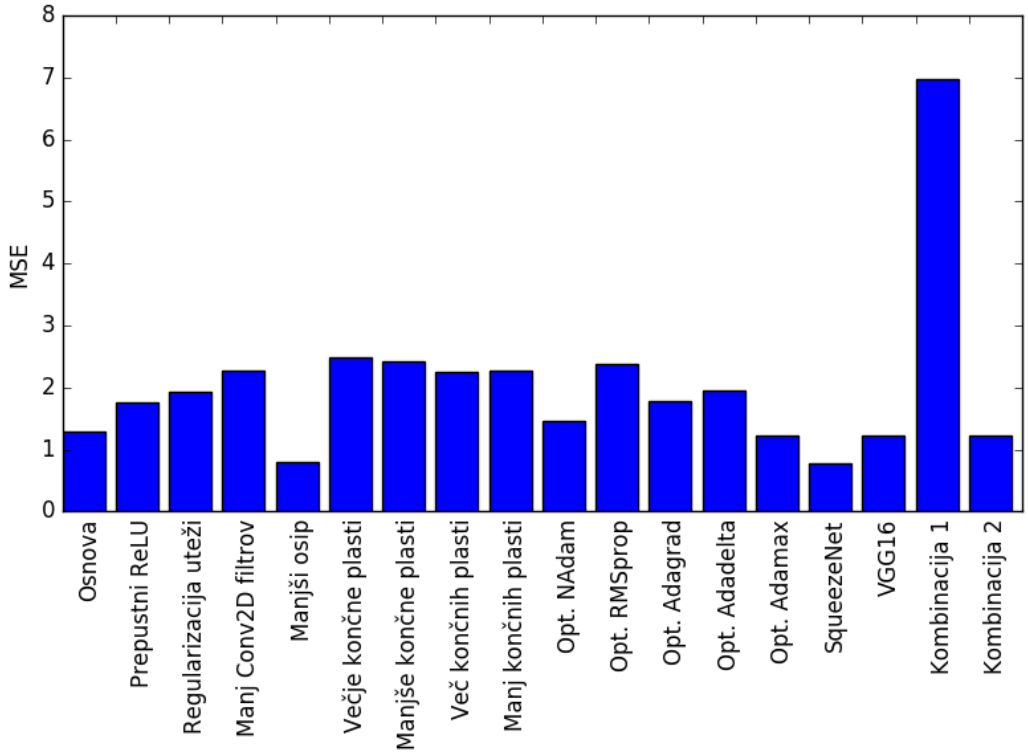
Na Sliki 4.4 smo prikazali tudi končno povprečno kvadratno napako na testni množici za različne variacije mreže. Rezultati so pretežno v skladu z ugotovitvami iz prejšnjih grafov. Če bi se pojavila kakšna velika odstopanja, bi lahko zaključili, da smo mrežo prekomerno umerili na učno množico.

4.0.2 Segmentacija

Podatkovna baza za FRCNN je bila sestavljena iz 1000 učnih in 100 testnih scen. Tudi tu je bila vsaka scena predstavljena s šestimi globinskimi slikami. V vsako sceno smo naključno razpostavili do tri superkvadrike. Na Sliki 4.5 je prikazana povprečna napaka parametrov in položaja ter oblike regij po vsaki iteraciji učenja. Povprečno napako smo računali kot evklidsko razdaljo med vektorjema napovedi in pravilnih vrednosti po sledeči enačbi

$$d = \frac{\sum_{j=1}^{N} \sqrt{\sum_{i=1}^{n} (y_i - p_i)^2}}{N}. \tag{4.2}$$

Tako kot pri povprečni kvadratni napaki N predstavlja število napovedanih vektorjev vsako iteracijo, n dimenzijo oz. število komponent vektorja, y_i i-to komponento napovedanega vektorja in p_i i-to komponenta pravilnega



Slika 4.4: Povprečna kvadratna napaka na testni množici za različne variacije nevronske mreže.

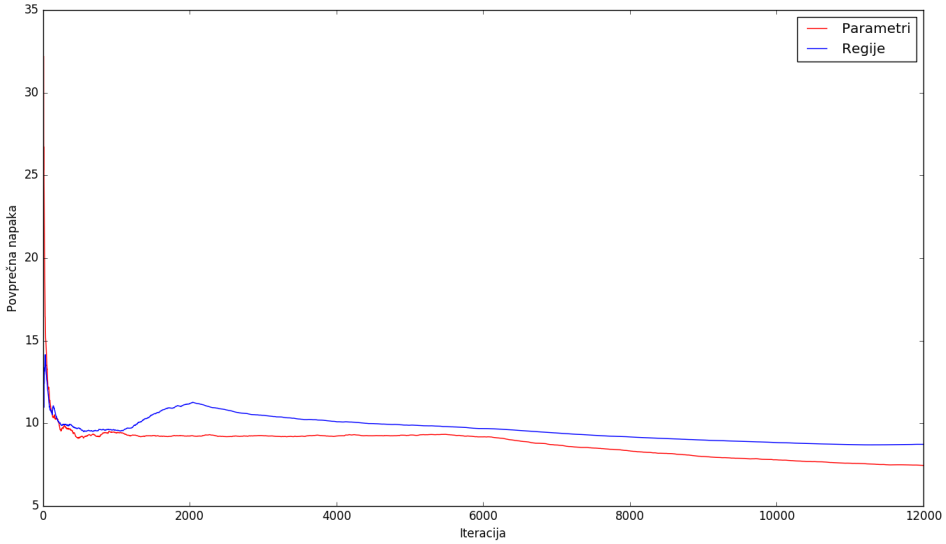
vektorja. Vektorji parametrov so bili dimenzije $n = 6$ (šest parametrov superkvadrika), vektorji regij pa dimenzije $n = 4$ (x in y koordinati, širina in višina regije).

V vsaki iteraciji smo v mrežo podali eno globinsko sliko, da smo dobili regije. Nato smo regije združili v sklope (ang. batch) velikosti 6 in regresijski del učili na vseh najdenih regijah za sliko po sklopih. Velikost sklopa regij je neodvisna od števila generiranih globinskih slik na posamezno sceno. Prav tako je neodvisna od števila parametrov objektov. Velikost sklopa je bila izbrana pretežno arbitrarno. Glede na to, da je napovedanih regij mnogo več kot globinskih slik, smo želeli nekoliko zmanjšati vpliv regresijskega dela mreže na skupne plasti. Tako se šele z vsako šesto regijo popravijo uteži v mreži na podlagi povprečne napake za te regije, namesto da se popravlja uteži po vsaki regiji, ki gre skozi mrežo.

Mrežo smo učili 10 epoh 15,5 ur. Slika 4.5 prikazuje samo prvi dve epohi učenja, za tem se napaka ni več bistveno spreminjala.

Povprečna napaka po učenju nevronske mreže je bila za RPN $8,73$, za regresijo pa $7,45$. Po učenju smo mrežo še testirali na testni množici. Povprečna napaka, ki smo jo dobili za testno množico, je bila $8,94$ za RPN in $10,95$ za regresijo. Ti rezultati kažejo, da smo RPN pretežno pravilno naučili ocenjevati regije, regresijo smo pa morda malenkost preveč umerili na učno množico. Je pa tudi res, da je natančnost regresije zelo odvisna od natančnosti RPN. Glede na to da regresijo učimo na najdenih regijah, če regije ne vsebujejo objekta, potem regresija težko karkoli napove.

Testiranje mreže na 100 scenah je trajalo 245s. To pomeni, da je za procesiranje posamezne scene mreža potrebovala povprečno 2.45s oz. 0.4s na posamezno globinsko sliko. Verjamemo tudi, da bi se dalo še nekoliko optimizirati računanje regij iz končnih plasti RPN, kar bi še pohitrilo izvajanje. Poleg tega bi bilo možno paralelizirati regresijski del. Po tem ko se poračunajo regije, bi lahko za vsak objekt napovedali parametre hkrati paralelno. Glede na to da smo testirali scene, ki so povprečno vsebovale dva objekta, bi to lahko tudi nekoliko pohitrilo izvajanje. Največjo pohitritev bi



Slika 4.5: Povprečna napaka napovedi regij in parametrov po vsaki iteraciji učenja.

verjetno lahko dosegli s pred procesiranjem globinskih slik tako, da najdemo pogled, kjer so objekti najbolje ločeni med sabo. V tem primeru bi lahko vsako sceno predstavili samo z eno sliko namesto s šestimi.

Naše rezultate primerjamo z rezultati Duncan et al. [21], Solina et al. [22] ter Stopnišek [12] v Tabeli 4.1. Pri Stopnišku primerjamo naše rezultate z njihovimi rezultati dimenzij kvadrastih predmetov modela Sutivan.

Iz Tabele 4.1 lahko vidimo, da smo z našo metodo dosegli najboljšo parametrizacijo objektov v sceni, vendar smo morali za to nekoliko kompromizirati povprečni čas izvajanja.

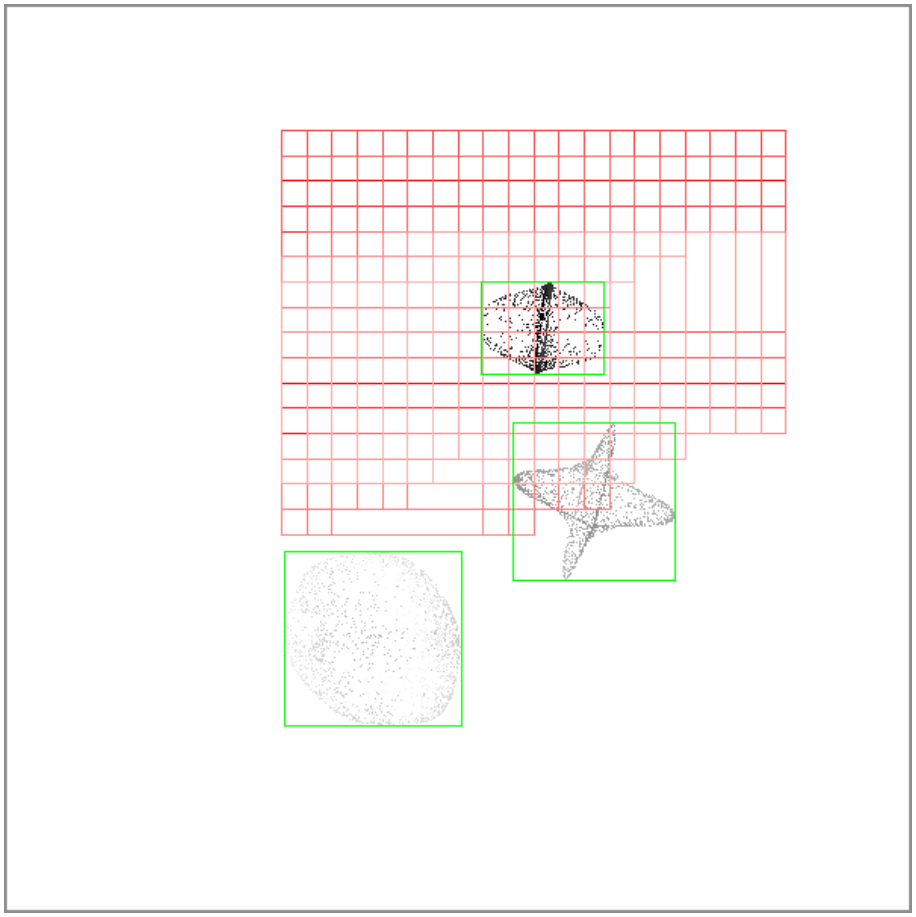
Slika 4.6 prikazuje primer najdenih regij za en objekt v sceni. Z zeleno so prikazane pravilne oznake za objekte, z rdečimi odtenki pa najdene regije. Na Sliki 4.7 so prikazane samo štiri od teh regij za boljšo preglednost. Vse regije, ki jih lahko najdemo z RPN-jem, imajo vnaprej določeno velikost, glede na to da uporabljamo sidra. Mi smo določili devet različnih velikosti sider, kot je opisano v poglavju 3.3.1. V tem primeru so vse regije istih velikosti,

Tabela 4.1: Primerjava rezultatov z Duncan et al. [21], Solina et al. [22] ter Stopnišek [12]

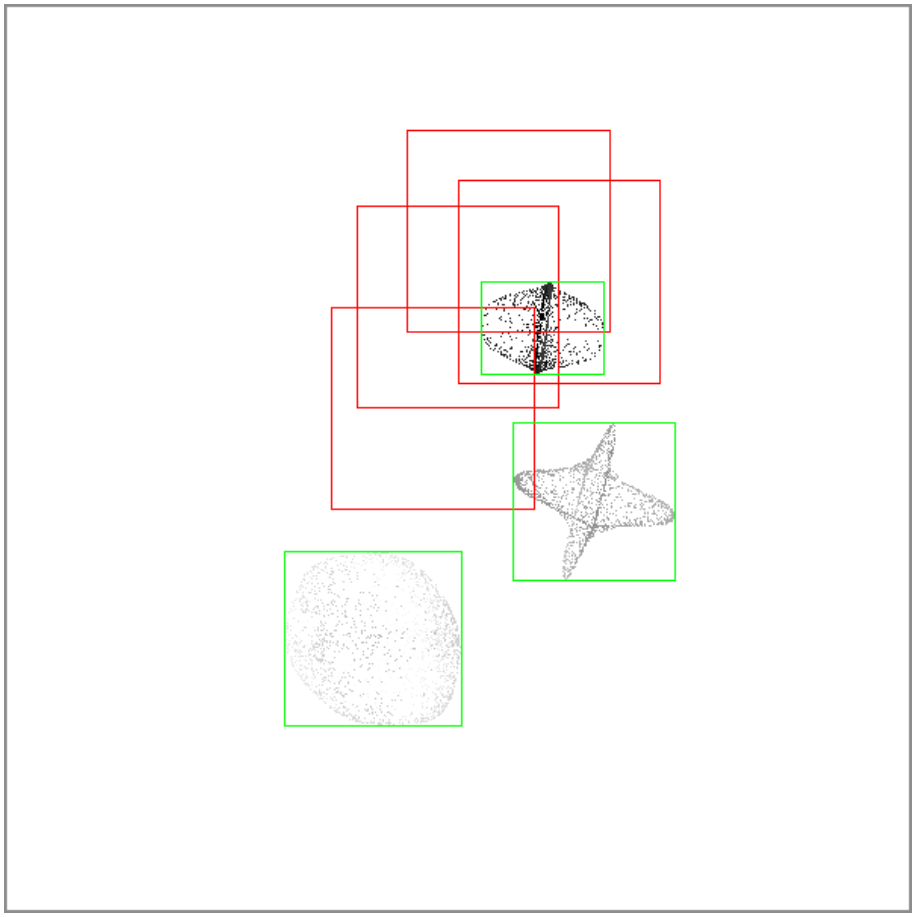
	Povprečna napaka	Povprečni čas (s)
Naša metoda	10,95	0.4
Duncan et al.	15,22	0.04
Solina et al.	14,1	2.1
Stopnišek	23,30	-

v splošnem primeru pa lahko dobimo različno velike regije na posamezen objekt.

Kar lahko opazimo iz teh slik je, da regij morda nismo dovolj dobro stisnili skupaj na vsak posamezen objekt. Iz tega najverjetneje dobimo tako veliko napako za natančnost RPN. Vsaka regija sicer ima nekaj prekrivanja s pravilno regijo in ima več prekrivanja z njo kot z regijami drugih objektov, vendar so predvsem robne regije še zmeraj precej prazne. Glede na to da imajo regije spodnjo mejo minimalnega dovoljenega prekrivanja, se samo zelo poredko lahko zgodi, da regija sploh ne vsebuje vsaj kosa objekta. Ta problem bi morda lahko rešili z boljšim zatiranjem vrednosti, ki niso maksimalne, ali z agregacijo regij. Bežno smo poskusili agregirati regije posameznih objektov skupaj v eno regijo na objekt, vendar so bile regije vedno toliko zamaknjene, da je regresija parametrov delala slabše. To kaže na to, da je bolje imeti nekaj dobrih in nekaj slabih regij za učenje regresije, kot pa samo eno regijo srednje kvalitete. Tako bo v povprečnem primeru regresija parametrov boljša.



Slika 4.6: Vse regije najdene z RPN za specifičen objekt.



Slika 4.7: Primeri regij najdenih z RPN.

Poglavje 5

Zaključek

V tem magistrskem delu smo implementirali nevronsko mrežo, s katero lahko segmentiramo in rekonstruiramo objekte iz 3D oblaka točk. Prva verzija mreže je bila zelo preprosta. Namenjena je bila samo napovedi parametrov enega objekta v sceni. Nato smo arhitekturo nadgradili z mrežo za iskanje predlogov regij in celo mrežo razširili na nekoliko prirejen FRCNN. Za skupne plasti smo uporabili arhitekturo SqueezeNet. Le-ta se je skupaj z VGG16 najbolj izkazala pri prvi verziji mreže, vendar jo je za razliko od VGG16 moč mnogo hitreje učiti, glede na to da je manjša. S pridobljenimi rezultati smo pokazali, da je z uporabo nevronskih mrež možno doseči visoko natančnost pri parametrizaciji objektov in to parametrizacijo opraviti v relativno realnem času. Prednost uporabe nevronskih mrež je tudi neodvisnost od kompleksnosti scene. Z drugimi metodami se praviloma časovna kompleksnost povečuje s številom točk v oblaku, z našo metodo je pa vseeno kako gost je oblak točk, saj je časovna zahtevnost odvisna predvsem od resolucije globinskih slik.

Uporaba nevronskih mrež za reševanje takih problemov se je v zadnjih nekaj letih precej razširila. Glede na to da senzorji za zajem vizualnih informacij postajajo vedno bolj zmogljivi, je potrebno te podatke vse bolj učinkovito obdelati. Nevronske mreže omogočajo relativno hitro procesiranje podatkov, v primerjavi z drugimi metodami npr. prileganje geometrijskih teles na oblake

točk. En od primerov uporabe je npr. pri orientaciji kamere in kartiranju prostora [17].

V nadaljnjem delu bi lahko poskusili še druge arhitekture za skupne plasti naše mreže, npr. ResNet. Prav tako bi lahko poskusili ločiti skupne plasti, da bi dobili mrežo za napoved predlogov regij in ločeno mrežo za napoved parametrov v regijah. Tako učenje napovedi parametrov zagotovo ne bi vplivalo na učenje predlogov regij in obratno, bi pa s tem seveda povečali časovno zahtevnost. Pametno bi bilo preizkustiti tudi kakšen boljši način agregacije predlaganih regij. Verjetno največja sprememba, ki bi jo lahko poskusili, je drugačna predstavitev oblaka točk. Namesto globinskih slik, bi lahko uporabili vokselizacijo scene. Za to bi bilo treba precej spremeniti kodo, da namesto 2D slik mreža obdeluje 3D voksle. Verjamemo, da bi se dalo dosežene rezultate sčasoma še izboljšati.

Dodatek A

Variacije preproste mreže za napoved parametrov

Tu so naštete različne variacije prvotne preproste konvolucijske mreže, katere smo testirali. Koda je v jeziku Python, namenjena za uporabo s knjižnico Keras.

Različica A.1: Osnovna mreža, ki smo jo uporabili za izhodišče vseh nadaljnih sprememb.

```
model.add(Conv2D(32, kernel_size=(3, 3),
          activation='relu', input_shape=(rows,cols,1)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_params, activation='linear'))
opt = Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
           epsilon=1e-08, decay=0.0)
```

```
model.compile(loss='mean_squared_error', optimizer=opt,
              metrics=['mean_squared_error'])
```

Različica A.2: Prepustne ReLU aktivacije namesto navadnih ReLU aktivacijskih funkcij.

```
model.add(Conv2D(32, kernel_size=(3, 3),
          activation='relu', input_shape=(rows,cols,1)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='linear'))
model.add(LeakyReLU(alpha=.001))
model.add(Dense(128, activation='linear'))
model.add(LeakyReLU(alpha=.001))
model.add(Dense(128, activation='linear'))
model.add(LeakyReLU(alpha=.001))
model.add(Dropout(0.5))
model.add(Dense(num_params, activation='linear'))
opt = Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
           epsilon=1e-08, decay=0.0)
model.compile(loss='mean_squared_error', optimizer=opt,
              metrics=['mean_squared_error'])
```

Različica A.3: Regularizacija uteži.

```
model.add(Conv2D(32, kernel_size=(3, 3),
          activation='relu', input_shape=(rows,cols,1)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
```

```
model.add(Dense(128, activation='relu',
          kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(128, activation='relu',
          kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(128, activation='relu',
          kernel_regularizer=regularizers.l2(0.01)))
model.add(Dropout(0.5))
model.add(Dense(num_params, activation='linear'))
opt = Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
           epsilon=1e-08, decay=0.0)
model.compile(loss='mean_squared_error', optimizer=opt,
              metrics=['mean_squared_error'])
```

Različica A.4: Manj konvolucijskih filtrov.

```
model.add(Conv2D(16, kernel_size=(3, 3),
          activation='relu', input_shape=(rows,cols,1)))
model.add(Conv2D(16, (3, 3), activation='relu'))
model.add(Conv2D(16, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_params, activation='linear'))
opt = Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
           epsilon=1e-08, decay=0.0)
model.compile(loss='mean_squared_error', optimizer=opt,
              metrics=['mean_squared_error'])
```

Različica A.5: Manjši izpust.

```
model.add(Conv2D(32, kernel_size=(3, 3),
```

```
            activation='relu', input_shape=(rows,cols,1)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.125))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(num_params, activation='linear'))
opt = Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
            epsilon=1e-08, decay=0.0)
model.compile(loss='mean_squared_error', optimizer=opt,
                metrics=['mean_squared_error'])
```

Različica A.6: Povečane končne polno povezane plasti.

```
model.add(Conv2D(32, kernel_size=(3, 3),
            activation='relu', input_shape=(rows,cols,1)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_params, activation='linear'))
opt = Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
            epsilon=1e-08, decay=0.0)
model.compile(loss='mean_squared_error', optimizer=opt,
                metrics=['mean_squared_error'])
```

Različica A.7: Zmanjšane končne polno povezane plasti.

```
model.add(Conv2D(32, kernel_size=(3, 3),
          activation='relu', input_shape=(rows,cols,1)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_params, activation='linear'))
opt = Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
           epsilon=1e-08, decay=0.0)
model.compile(loss='mean_squared_error', optimizer=opt,
              metrics=['mean_squared_error'])
```

Različica A.8: Več končnih polno povezanih plasti.

```
model.add(Conv2D(32, kernel_size=(3, 3),
          activation='relu', input_shape=(rows,cols,1)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_params, activation='linear'))
opt = Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
```

```
            epsilon=1e-08, decay=0.0)
model.compile(loss='mean_squared_error', optimizer=opt,
              metrics=['mean_squared_error'])
```

Različica A.9: Manj končnih polno povezanih plasti.

```
model.add(Conv2D(32, kernel_size=(3, 3),
          activation='relu', input_shape=(rows,cols,1)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_params, activation='linear'))
opt = Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
           epsilon=1e-08, decay=0.0)
model.compile(loss='mean_squared_error', optimizer=opt,
              metrics=['mean_squared_error'])
```

Različica A.10: Prva kombinacija več variacij. Manj konvolucijskih filtrov v zadnji Conv2D plasti, normalizacija, dodatna nekoliko manjša polno povezana plast, regularizacija uteži in manjši končni izpust.

```
model.add(Conv2D(32, kernel_size=(3, 3),
          activation='relu', input_shape=(rows,cols,1)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(Conv2D(16, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(64, activation='tanh',
```

```
            kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(128, activation='tanh',
            kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(128, activation='tanh',
            kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(64, activation='tanh',
            kernel_regularizer=regularizers.l2(0.01)))
model.add(Dropout(0.25))
model.add(Dense(num_params, activation='linear'))
opt = Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
                    epsilon=1e-08, decay=0.0)
model.compile(loss='mean_squared_error', optimizer=opt,
                metrics=['mean_squared_error'])
```

Različica A.11: Druga kombinacija več variacij. Normalizacija, dodatna polno povezana plast, povečanje vseh polno povezanih plasti, regularizacija uteži in manjši končni izpust.

```
model.add(Conv2D(32, kernel_size=(3, 3),
            activation='relu', input_shape=(rows,cols,1)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(256, activation='tanh',
            kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(256, activation='tanh',
            kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(256, activation='tanh',
            kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(256, activation='tanh',
            kernel_regularizer=regularizers.l2(0.01)))
```

```
model.add(Dropout(0.25))
model.add(Dense(num_params, activation='linear'))
opt = Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
           epsilon=1e-08, decay=0.0)
model.compile(loss='mean_squared_error', optimizer=opt,
              metrics=['mean_squared_error'])
```

Literatura

[1] A. Jackson, M. Valstar, G. Tzimiropoulos, A CNN cascade for landmark guided semantic part segmentation, arXiv preprint arXiv:1609.09642.

[2] E. Grant, P. Kohli, M. van Gerven, Deep disentangled representations for volumetric reconstruction, arXiv preprint arXiv:1610.03777.

[3] I. Biederman, Recognition-by-components: a theory of human image understanding., Psychological review 94 (2) (1987) 115.

[4] J. Ponce, D. Chelberg, Finding the limbs and cusps of generalized cylinders, International Journal of Computer Vision 1 (3) (1988) 195–210.

[5] G. Sithole, G. Vosselman, Automatic structure detection in a point-cloud of an urban landscape, in: Remote Sensing and Data Fusion over Urban Areas, 2003. 2nd GRSS/ISPRS Joint Workshop on, IEEE, 2003, pp. 67–71.

[6] Q.-Y. Zhou, U. Neumann, Complete residential urban area reconstruction from dense aerial LiDAR point clouds, Graphical Models 75 (3) (2013) 118–125.

[7] R. Schnabel, R. Wahl, R. Klein, Efficient RANSAC for point-cloud shape detection, in: Computer graphics forum, Vol. 26, Wiley Online Library, 2007, pp. 214–226.

[8] R. B. Rusu, N. Blodow, Z. C. Marton, M. Beetz, Close-range scene segmentation and reconstruction of 3D point cloud maps for mobile ma-

nipulation in domestic environments, in: 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2009, pp. 1–6.

[9] S. Song, J. Xiao, Sliding shapes for 3D object detection in depth images, in: European Conference on Computer Vision, Springer, 2014, pp. 634–651.

[10] D. Sedlacek, J. Zara, Graph cut based point-cloud segmentation for polygonal reconstruction, in: International Symposium on Visual Computing, Springer, 2009, pp. 218–227.

[11] A. Leonardis, A. Jaklič, F. Solina, Superquadrics for segmenting and modeling range data, IEEE Transactions on Pattern Analysis and Machine Intelligence 19 (11) (1997) 1289–1295.

[12] Ž. Stopinšek, Segmentacija in rekonstrukcija kulturne dediščine iz fotogrametrično pridobljenega oblaka točk, Ph.D. thesis, Univerza v Ljubljani (2016).

[13] M. Ozkan, B. M. Dawant, R. J. Maciunas, Neural-network-based segmentation of multi-modal medical images: a comparative and prospective study, IEEE transactions on Medical Imaging 12 (3) (1993) 534–544.

[14] H. Masoumi, A. Behrad, M. A. Pourmina, A. Roosta, Automatic liver segmentation in MRI images using an iterative watershed algorithm and artificial neural network, Biomedical Signal Processing and Control 7 (5) (2012) 429–437.

[15] A. Sharma, O. Grau, M. Fritz, Vconv-dae: Deep volumetric shape learning without object labels, arXiv preprint arXiv:1604.03755.

[16] A. Salehi, V. Gay-bellile, S. Bourgeois, F. Chausse, Improving constrained bundle adjustment through semantic scene labeling, in: Geometry Meets Deep Learning ECCV 2016 Workshop, 2016.

[17] T. Cavallari, L. Di Stefano, On-line large scale semantic fusion, in: Geometry Meets Deep Learning ECCV 2016 Workshop, 2016.

[18] S. Park, J. Hwang, N. Kwak, 3D human pose estimation using convolutional neural networks with 2D pose information, arXiv preprint arXiv:1608.03075.

[19] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, K. Keutzer, SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1mb model size, CoRR abs/1602.07360.
URL `http://arxiv.org/abs/1602.07360`

[20] S. Ren, K. He, R. B. Girshick, J. Sun, Faster R-CNN: towards real-time object detection with region proposal networks, CoRR abs/1506.01497.
URL `http://arxiv.org/abs/1506.01497`

[21] K. Duncan, S. Sarkar, R. Alqasemi, R. Dubey, Multi-scale superquadric fitting for efficient shape and pose recovery of unknown objects, in: Robotics and Automation (ICRA), 2013 IEEE International Conference on, IEEE, 2013, pp. 4238–4243.

[22] F. Solina, R. Bajcsy, Recovery of parametric models from range images: the case for superquadrics with global deformations, IEEE Transactions on Pattern Analysis and Machine Intelligence 12 (2) (1990) 131–147. `doi:10.1109/34.44401`.